

Lo stato

- ▶ Insieme di associazioni tra **nomi** e **valori**.
 $x \rightsquigarrow 5$
- ▶ Nelle specifiche, si vogliono spesso indicare valori costanti ma **generici**. Utilizziamo le seguenti **convenzioni**:
 - ▶ usiamo lettere minuscole per i **nomi simbolici** utilizzati nello stato e negli algoritmi;
 - ▶ usiamo lettere maiuscole per indicare **costanti generiche**
- ▶ Laddove necessario, possiamo specificare condizioni al contorno sul dominio di appartenenza di tali costanti.

Esempio:

$\{\text{naturale} \rightsquigarrow A, \text{base} \rightsquigarrow 2\} \quad \text{con } A \geq 0$

Al nome simbolico **naturale** è associato un generico valore naturale ($A \geq 0$), mentre al nome simbolico **base** è associata la costante 2.

ATTENZIONE: all'interno degli algoritmi non è possibile utilizzare esplicitamente le costanti generiche.

Esempio: Elevamento a potenza

Specifica:

Stato iniziale: $\{\text{base} \rightsquigarrow A, \text{esponente} \rightsquigarrow B\} \quad \text{con } A > 0 \text{ e } B \geq 0$

Stato finale: $\{\text{risultato} \rightsquigarrow A^B\}$

Algoritmo:

```
risultato = 1;
while (esponente > 0) {
    risultato = risultato * A;      NO!
    esponente = esponente - 1;
}
```

Le espressioni non possono contenere costanti **generiche** (proprio perché tali!).

Esempi

Esempio: Ordinare tre valori interi distinti tra loro

Stato iniziale: $\{x \rightsquigarrow A, y \rightsquigarrow B, z \rightsquigarrow C\}$ con A, B, C distinti tra loro

Stato finale: $\{x \rightsquigarrow \max(\{A, B, C\}),$
 $y \rightsquigarrow \max(\{A, B, C\} \setminus \{\max(\{A, B, C\})\}),$
 $z \rightsquigarrow \min(\{A, B, C\})\}$

Algoritmo

Passo 1. “Ordiniamo” x e y , portandoci nello stato intermedio

Stato 1: $\{x \rightsquigarrow \max(A, B), y \rightsquigarrow \min(A, B), z \rightsquigarrow C\}$

Passo 2. “Ordiniamo” x e z , portandoci nello stato intermedio

Stato 2: $\{x \rightsquigarrow \max(\{A, B, C\}), y \rightsquigarrow \min(A, B),$
 $z \rightsquigarrow \min(\max(A, B), C)\}$

Passo 3. “Ordiniamo” y e z , portandoci nello stato finale desiderato

Soluzione nello pseudo-linguaggio

```

if ( $x < y$ )
{
    temp = x;
    x = y;
    y = temp;
}
else ;
     $\{x \rightsquigarrow \max(A, B), y \rightsquigarrow \min(A, B), z \rightsquigarrow C\}$ 
if ( $x < z$ )
{
    temp = x;
    x = z;
    z = temp;
}
else ;
     $\{x \rightsquigarrow \max(\{A, B, C\}), y \rightsquigarrow \min(A, B), z \rightsquigarrow \min(\max(A, B), C)\}$ 
if ( $y < z$ )
{
    temp = y;
    y = z;
    z = temp;
}
else ;
    Stato finale

```

Esempio: Calcolo del fattoriale di un numero naturale.
Ricordiamo che:

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ 1 \cdot 2 \cdot \dots \cdot (n-1) \cdot n & \text{se } n > 0 \end{cases}$$

Specifica:

Stato iniziale: $\{n \rightsquigarrow N\}$ con $N \geq 0$

Stato finale: $\{n \rightsquigarrow N, \text{fatt} \rightsquigarrow N!\}$

- Attenzione: la specifica impone che il valore di n non deve cambiare!

Soluzione:

```
fatt = 1;
i = 1;
while (i <= n)
{
    fatt = fatt * i;
    i = i + 1;
}
```

- i viene spesso chiamata **variabile di controllo** del ciclo, dal momento che l'esecuzione di ogni iterazione dipende dal valore di i .

Viene incrementata di 1 alla fine di ogni iterazione

\Rightarrow per ogni valore di $i \in [1, N]$

- Ad ogni iterazione del ciclo, vale la seguente proprietà:

$$\text{fatt} = \prod_{j=1}^{i-1} j = (i-1)! \quad (\bullet)$$

- Al termine del ciclo, quando cioè $i=N+1$, la (\bullet) diventa $\text{fatt} = N!$, ovvero lo stato finale desiderato.

Sequenze: nomi simbolici con indice

Consentiamo anche l'uso di nomi simbolici con **indice**, per rappresentare sequenze.

$$c[i] \rightsquigarrow 5$$

dove i è un valore naturale. Ad esempio:

$$\{i \rightsquigarrow K, c[0] \rightsquigarrow 0, \dots, c[K-1] \rightsquigarrow 0\} \quad K \geq 0$$

Nello stato sono presenti:

- un'associazione per il nome simbolico i , al quale è associato un valore naturale K .
- K associazioni per i nomi simbolici $c[0], c[1], \dots, c[K-1]$, a ciascuno dei quali è associato il valore 0 .

All'interno degli algoritmi, consentiamo di riferire nomi simbolici con indice nella forma $c[exp]$ dove exp deve essere una espressione a valori **naturali**.

Anche in questo caso exp non deve contenere costanti generiche (nell'esempio, non è lecito un assegnamento del tipo $c[K-1] = 5$, mentre è lecito $c[i-1] = 5$).

Esempio: Calcolare il numero di elementi pari in una sequenza data.

Stato iniziale: $\{dim \rightsquigarrow K, c[0] \rightsquigarrow V_1, \dots, c[K-1] \rightsquigarrow V_{K-1}\}$ con $K > 0$

Stato finale: $\{count \rightsquigarrow \#\{j \mid j \in [0, K-1] \wedge V_j \text{ pari}\}\}$

- Dobbiamo calcolare

$$\sum_{\substack{0 \leq j \leq dim-1 \\ c[j] \text{ pari}}} 1$$

- Di nuovo, utilizziamo un ciclo controllato da una variabile di controllo i che assume tutti i valori nell'intervallo $[0, dim-1]$.
- Facciamo in modo che, ad ogni iterazione, valga la seguente proprietà

$$count = \sum_{\substack{0 \leq j \leq i-1 \\ c[j] \text{ pari}}} 1$$

- Se, alla fine del ciclo, $i=dim$, abbiamo quanto desiderato e cioè

$$count = \sum_{\substack{0 \leq j \leq dim-1 \\ c[j] \text{ pari}}} 1$$

Soluzione:

```

count = 0;
i = 0;                                punto 1
while (i < dim)
{
    if (c[i] % 2 == 0)
        count = count + 1;
    else ;
    i = i + 1;
}

```

- Nello stato iniziale del **while**, al punto (1), $\text{count} \rightsquigarrow 0$, $i \rightsquigarrow 0$ e dunque

$$\sum_{\substack{1 \leq j \leq i-1 \\ c_j \text{ pari}}} 1 = \sum_{\substack{0 \leq j \leq -1 \\ c_j \text{ pari}}} 1 = 0 = \text{count}$$

- il corpo del **while** mantiene vera la proprietà

$$\text{count} = \sum_{\substack{0 \leq j \leq i-1 \\ c_j \text{ pari}}} 1$$

Considerazioni generali

- In molti problemi è necessario operare allo stesso modo su tutti gli elementi di un intervallo dato

per ogni $j \in [a, b]$ OP_j

esempio: $\sum_{j=a}^b \exp_j$

- in tutti questi casi si ricorre a cicli in cui una variabile di controllo permette di **scandire** tutto l'intervallo di interesse

```

i = a;
while (i <= b)
{
    <operazione in funzione di i>
    i = i+1;
}

```

Esercizi proposti

Problema 1: Calcolare il numero di occorrenze di un valore dato in una sequenza data

Stato iniziale: $\{\text{dim} \rightsquigarrow K, \text{val} \rightsquigarrow V, c[0] \rightsquigarrow V_0, \dots, c[K-1] \rightsquigarrow V_{K-1}\}$ con $K > 0$

Stato finale: $\{\text{occ} \rightsquigarrow \#\{j \mid j \in [0, K-1] \wedge V_j = V\}\}$

Problema 2: Calcolare il massimo e il minimo di una sequenza data di interi

Stato iniziale: $\{\text{dim} \rightsquigarrow K, c[0] \rightsquigarrow V_0, \dots, c[K-1] \rightsquigarrow V_{K-1}\}$ con $K > 0$

Stato finale: $\{\text{massimo} \rightsquigarrow \max(\{V_0, \dots, V_{K-1}\}),$
 $\text{minimo} \rightsquigarrow \min(\{V_0, \dots, V_{K-1}\})\}$

Problema 3: Calcolare il numero di occorrenze del valore massimo in una sequenza di interi

Stato iniziale: ? Stato finale: ?

Problema 4: Calcolare il numero di occorrenze di una cifra nella rappresentazione decimale di un numero naturale

Soluzione problema 1:

Stato iniziale: $\{\text{dim} \rightsquigarrow K, \text{val} \rightsquigarrow V, c_0 \rightsquigarrow V_0, \dots, c_{K-1} \rightsquigarrow V_{K-1}\}$ con $K > 0$

Stato finale: $\{\text{occ} \rightsquigarrow \#\{j \mid j \in [0, K-1] \wedge V_j = V\}\}$

```
occ = 0;
i = 0;
while (i < dim)
{
    if (c[i] == val)
        occ = occ + 1;
    else ;
    i = i + 1;
}
```

Ad ogni iterazione del ciclo vale la proprietà:

$\text{occ} = \#\{j \mid j \in [0, i) \wedge c[j] = \text{val}\}$

Soluzione problema 2

Stato iniziale: $\{\text{dim} \rightsquigarrow K, c_0 \rightsquigarrow V_0, \dots, c_{K-1} \rightsquigarrow V_{K-1}\}$ con $K > 0$

Stato finale: $\{\text{massimo} \rightsquigarrow \max(\{V_0, \dots, V_{K-1}\}),$
 $\text{minimo} \rightsquigarrow \min(\{V_0, \dots, V_{K-1}\})\}$

- Preoccupiamoci prima di calcolare il **massimo**
- Scorriamo l'intera sequenza, attraverso una variabile di controllo **i**, facendo in modo che, ad ogni scansione, valga la proprietà

$$\text{massimo} = \max\{c[j] \mid j \in [0, i)\}$$

```

massimo = c[0];
i = 1;
while (i < dim)
{
    qui massimo = max{c[0], ..., c[i-1]}
    if (c[i] > massimo)
        massimo = c[i];
    else ;
    i = i + 1; anche qui massimo = max{c[0], ..., c[i-1]}
}

```

- Il calcolo del minimo è analogo. Otteniamo:

```

massimo = c[0];
minimo = c[0];
i = 1;
while (i < dim)
{
    if (c[i] > massimo)           calcolo del massimo
        massimo = c[i];
    else ;
    if (c[i] < minimo)           calcolo del minimo
        minimo = c[i];
    else ;
    i = i + 1;
}

```

- Usiamo un algoritmo piu' efficiente: diminuiamo il numero di confronti.

```

massimo = c[0];
minimo = c[0];
i = 1;
while (i < dim)
{
    if (c[i] > massimo)           calcolo del massimo
        massimo = c[i];
    else
        if (c[i] < minimo)       calcolo del minimo
            minimo = c[i];
        else ;
    i = i + 1;
}

```

Soluzione problema 3

Stato iniziale: $\{ \text{dim} \rightsquigarrow K, c[0] \rightsquigarrow V_0, \dots, c_{K-1} \rightsquigarrow V_{K-1} \}$

Stato finale: $\{ \text{occ} \rightsquigarrow \# \{ j \mid j \in [0, K-1] \wedge V_j = \max \{ V_0, \dots, V_{K-1} \} \}$

- Un algoritmo ingenuo:
 - Calcoliamo il valore massimo come nel problema 2
 - Scandiamo nuovamente la sequenza per contare il numero di occorrenze del massimo
- Comporta una duplice scansione della sequenza
- Un algoritmo che comporta una singola scansione si ottiene adattando quello visto per il calcolo del massimo


```

massimo = c[0];
i = 1;
occ = 1;
while (i < dim)
{
    if (c[i] > massimo)
    {
        massimo = c[i];
        occ = 1;
    }
    else
        if (c[i] == massimo)
            occ = occ + 1;
        else ;
    i = i + 1;
}

```

Soluzione problema 4

Stato iniziale: $\{\text{numero} \rightsquigarrow N, \text{cif} \rightsquigarrow C\}$ con $N > 0$

Stato finale: $\{\text{occ} \rightsquigarrow \#\{j \mid j \in [0, K] \wedge C_j = C\}\}$
dove $N = (C_K \dots C_0)_{10}$

- Analogo al problema 1, ma questa volta dobbiamo anche calcolare gli elementi della sequenza
- La lunghezza della sequenza non è nota a priori

```

n = numero; occ = 0;
while (n != 0)
{
    nuova_cifra = n % 10;
    if (nuova_cifra == cif)
        occ = occ + 1;
    else ;
    n = n / 10;
}

```

- questa volta la variabile di controllo è n

- ▶ Variante: **acquisire** un numero naturale ed una cifra decimale, contare il numero di occorrenze di quest'ultima nella rappresentazione decimale del numero letto, e produrlo in uscita.

```
Input(numero);
Input(cif);
n = numero; occ = 0;
while (n != 0)
{
    ...
    if (nuova_cifra == cif) ...
}
Output(occ);
```

- ▶ da uno stato iniziale che non contiene associazioni, o meglio su cui non facciamo alcuna ipotesi, ci portiamo in uno stato che corrisponde allo stato iniziale del problema precedente
- ▶ nello stato finale, produciamo in uscita il risultato calcolato

Rappresentazione binaria

- ▶ Per informazione intendiamo tutto quello che viene manipolato da un calcolatore:
 - ▶ numeri (naturali, interi, reali, ...)
 - ▶ caratteri
 - ▶ immagini
 - ▶ suoni
 - ▶ programmi
 - ▶ ...
- ▶ La più piccola unità di informazione memorizzabile o elaborabile da un calcolatore, il **bit**, corrisponde allo stato di un dispositivo fisico che viene interpretato come **1** o **0**.
- ▶ In un calcolatore tutte le informazioni sono rappresentate in **forma binaria**, come sequenze di **0** e **1**.
- ▶ Per **motivi tecnologici**: distinguere tra due valori di una grandezza fisica è più semplice che non ad esempio tra dieci valori.

Rappresentazione di numeri naturali

- ▶ Un numero naturale è un oggetto matematico, che può essere **rappresentato** mediante una **sequenza di simboli** di un alfabeto fissato.
- ▶ È importante distinguere tra numero e sua rappresentazione: il **numerales** "234" è la rappresentazione del **numero** 234.
- ▶ Si distinguono **2 tipi di rappresentazione**:
 - additiva**: ad es. le cifre romane
 - posizionale**: una cifra contribuisce con un valore diverso al numero a seconda della posizione in cui si trova
- ▶ Noi consideriamo solo la rappresentazione posizionale.

Rappresentazione posizionale

- ▶ Un numero è rappresentato da una **sequenza finita di cifre** di un certo **alfabeto**:

$$c_{n-1}c_{n-2} \cdots c_1c_0 = N_b$$

c_0 viene detta cifra **meno significativa**

c_{n-1} viene detta cifra **più significativa**

- ▶ Il numero b di cifre diverse (dimensione dell'alfabeto) è detto **base** del sistema di numerazione.
- ▶ Ad ogni cifra è associato un valore compreso tra 0 e $b - 1$.

Base	Alfabeto	Sistema
2	0, 1	binario
8	0, ..., 7	ottale
10	0, ..., 9	decimale
16	0, ..., 9, A, ..., F	esadecimale

- Il significato di una sequenza di cifre (il numero N che essa rappresenta) dipende dalla base b :

$$c_{n-1} \cdot b^{n-1} + c_{n-2} \cdot b^{n-2} + \dots + c_1 \cdot b^1 + c_0 \cdot b^0 = \sum_{i=0}^{n-1} c_i \cdot b^i = N$$

Esempio: Il numerale 101 rappresenta numeri diversi a seconda del sistema usato:

Sistema	Base b	101_b	Valore ₁₀
decimale	10	$(101)_{10}$	101
binario	2	$(101)_2$	5
ottale	8	$(101)_8$	65
esadecimale	16	$(101)_{16}$	257

Intervallo di rappresentazione

- I numeri rappresentabili in base b con n posizioni (cifre) vanno da 0 a $b^n - 1$.

3 cifre in base 10 : da	0	a	$999 = 10^3 - 1$
8 cifre in base 2 : da	0	a	$255 = 2^8 - 1$
16 cifre in base 2 : da	0	a	$65\,535 = 2^{16} - 1$
32 cifre in base 2 : da	0	a	$4\,294\,967\,296 = 2^{32} - 1$
2 cifre in base 16 : da	0	a	$255 = 16^2 - 1$
8 cifre in base 16 : da	0	a	$4\,294\,967\,296 = 16^8 - 1$

Conversioni di base: da base b a base 10

- Usando direttamente

$$c_{n-1} \cdot b^{n-1} + c_{n-2} \cdot b^{n-2} + \dots + c_1 \cdot b^1 + c_0 \cdot b^0 = \sum_{i=0}^{n-1} c_i \cdot b^i = N$$

esprimendo le cifre e b in base 10 (e facendo i conti in base 10)

Esercizio

(domani) Scrivere l'algoritmo di conversione da base b a base 10.

Conversioni di base: da base 10 a base b

$$\begin{aligned} N &= c_0 + c_1 \cdot b^1 + c_2 \cdot b^2 + \dots + c_{k-1} \cdot b^{k-1} \\ &= c_0 + b \cdot (c_1 + b \cdot (c_2 + \dots + b \cdot c_{k-1}) \dots) \end{aligned}$$

- Vogliamo determinare le cifre c_0, c_1, \dots, c_{k-1}
- Consideriamo la divisione di N per b :

$$\begin{aligned} N &= R + b \cdot Q \quad (0 \leq R < b) \\ &= c_0 + b \cdot (c_1 + b \cdot (\dots)) \end{aligned}$$

↓

$$R = c_0 \quad \text{ovvero, il resto } R \text{ della divisione di } N \text{ per } b \text{ dà}$$

$$c_0 \quad (\text{cifra meno significativa})$$

$$Q = c_1 + b \cdot (\dots)$$

- A partire dal quoziente Q si può iterare il procedimento per ottenere le cifre successive (fino a che Q diventa 0).

Conversione da base 10 a base b

```
i = 0;
while (num != 0) {
    c[i] = num % b;
    num = num / b;
    i = i+1; }
```

N.B. Le cifre vengono determinate dalla meno significativa alla più significativa.

Esempio: $(25)_{10} = (???)_2 = (11001)_2$

$N : b$	Q	R	cifra
$25 : 2$	12	1	c_0
$12 : 2$	6	0	c_1
$6 : 2$	3	0	c_2
$3 : 2$	1	1	c_3
$1 : 2$	0	1	c_4

N.B. servono 5 bit (con cui possiamo rappresentare i numeri da 0 a 31)

Rappresentazione di numeri interi

- Dobbiamo rappresentare anche il **segno**: si usa uno dei bit (quello più significativo)

Rappresentazione tramite modulo e segno

- il bit più significativo rappresenta il segno
 - le altre $n - 1$ cifre rappresentano il valore assoluto
 - problemi:
 - doppia rappresentazione per lo zero ($00 \dots 00$ e $10 \dots 00$)
 - le operazioni aritmetiche sono complicate (analisi per casi)
- ⇒ invece della rappresentazione tramite modulo e segno si usa una rappresentazione in **complemento alla base**

Rappresentazione in complemento alla base

In quanto segue b indica la base e n indica il numero complessivo di cifre.

- ▶ Con base b e n cifre, abbiamo a disposizione b^n configurazioni distinte.
- ▶ Utilizziamo metà delle configurazioni per rappresentare numeri positivi e l'altra metà per rappresentare numeri negativi.

$$||X|| = \begin{cases} X & \text{se } X \geq 0 \\ b^n - |X| & \text{se } X < 0 \end{cases}$$

- ▶ in questo modo si rappresentano gli interi relativi nell'intervallo $[-b^n/2, b^n/2)$
 - ▶ se $X \geq 0$: $||X||$ è compresa in $[0, b^n/2)$
 - ▶ se $X < 0$: $||X||$ è compresa in $[b^n/2, b^n)$
- ▶ lo 0 ha una sola rappresentazione

Rappresentazione in complemento alla base

N	$b = 10$ e $n = 1$	$b = 2$ e $n = 3$
-5	5	
-4	6	100
-3	7	101
-2	8	110
-1	9	111
0	0	000
1	1	001
2	2	010
3	3	011
4	4	

- ▶ se $b = 2 \Rightarrow$ rappresentazione in **complemento a 2**
 - ▶ rappresentazione degli interi relativi nell'intervallo $[-2^{n-1}, 2^{n-1})$
 - ▶ positivi: la cifra più significativa è 0 (rappresentati nella parte inferiore dell'intervallo)
 - ▶ negativi: la cifra più significativa è 1 (rappresentati nella parte superiore dell'intervallo)

Operazione di complementazione

Vogliamo determinare un algoritmo per determinare la rappresentazione in complemento alla base di $-X$, data quella di X . Indipendentemente dal segno di X , abbiamo:

$$||X|| + ||-X|| = |X| + b^n - |X| = b^n$$

per cui

$$||-X|| = b^n - ||X||$$

o equivalentemente

$$||-X|| = b^n - 1 - ||X|| + 1$$

Operazione di complementazione

► Supponiamo:

$$||X|| = \sum_{i=0}^{n-1} c_i \cdot b^i$$

e ricordiamo che la rappresentazione di $b^n - 1$ è

$$\sum_{i=0}^{n-1} (b-1) \cdot b^i$$

► Otteniamo:

$$\begin{aligned} ||-X|| &= b^n - 1 - ||X|| + 1 \\ &= \left(\sum_{i=0}^{n-1} (b-1) \cdot b^i \right) - \left(\sum_{i=0}^{n-1} c_i \cdot b^i \right) + 1 \end{aligned}$$

Operazione di complementazione

- Sia ora k la prima posizione significativa di $||X||$, ovvero la prima cifra (a partire da destra) diversa da 0. Abbiamo allora:

$$\begin{aligned}
 ||-X|| &= \left(\sum_{i=0}^{n-1} (b-1) \cdot b^i \right) - \left(\sum_{i=0}^{n-1} c_i \cdot b^i \right) + 1 \\
 &= \left(\sum_{i=0}^{n-1} (b-1) \cdot b^i \right) - \left(\sum_{i=k}^{n-1} c_i \cdot b^i \right) + 1 \\
 &= \left(\sum_{i=k+1}^{n-1} ((b-1) - c_i) \cdot b^i \right) + ((b-1) - c_k) \cdot b^k + \\
 &\quad \left(\sum_{i=0}^{k-1} (b-1) \cdot b^i \right) + 1
 \end{aligned}$$

- Osserviamo ora che $\left(\sum_{i=0}^{k-1} (b-1) \cdot b^i \right) + 1 = b^k$.

Operazione di complementazione

- Sia ora k la prima posizione significativa di $||X||$, ovvero la prima cifra (a partire da destra) diversa da 0. Abbiamo allora:

$$\begin{aligned}
 ||-X|| &= \left(\sum_{i=0}^{n-1} (b-1) \cdot b^i \right) - \left(\sum_{i=0}^{n-1} c_i \cdot b^i \right) + 1 \\
 &= \left(\sum_{i=0}^{n-1} (b-1) \cdot b^i \right) - \left(\sum_{i=k}^{n-1} c_i \cdot b^i \right) + 1 \\
 &= \left(\sum_{i=k+1}^{n-1} ((b-1) - c_i) \cdot b^i \right) + ((b-1) - c_k) \cdot b^k + b^k
 \end{aligned}$$

Operazione di complementazione

$$\begin{aligned} ||-X|| &= \left(\sum_{i=k+1}^{n-1} ((b-1) - c_i) \cdot b^i \right) + ((b-1) - c_k) \cdot b^k + b^k \\ &= \left(\sum_{i=k+1}^{n-1} ((b-1) - c_i) \cdot b^i \right) + (b - c_k) \cdot b^k \end{aligned}$$

- L'ultimo addendo è 0, poichè $c_i = 0$, per ogni $i = 0, \dots, k-1$.
- Come possiamo leggere quanto ottenuto?

La rappresentazione di $-X$ si ottiene da quella di X :

1. ricopiando gli zeri meno significativi
2. complementando alla base la primacifra significativa
3. complementando alla base meno uno le rimanenti cifre

Operazione di complementazione

$$\begin{aligned} ||-X|| &= \left(\sum_{i=k+1}^{n-1} ((b-1) - c_i) \cdot b^i \right) + ((b-1) - c_k) \cdot b^k + b^k \\ &= \left(\sum_{i=k+1}^{n-1} ((b-1) - c_i) \cdot b^i \right) + (b - c_k) \cdot b^k + \left(\sum_{i=0}^{k-1} c_i \cdot b^i \right) \end{aligned}$$

- L'ultimo addendo è 0, poichè $c_i = 0$, per ogni $i = 0, \dots, k-1$.
- Come possiamo leggere quanto ottenuto?

La rappresentazione di $-X$ si ottiene da quella di X :

1. ricopiando gli zeri meno significativi
2. complementando alla base la prima cifra significativa
3. complementando alla base meno uno le rimanenti cifre

Operazione di complementazione

Esempio: $b = 3$, $n = 4$, $||X|| = 0210$ (dunque $X = (21)_{10}$)

$$||-X|| = 2020$$

Verifichiamo:

- ▶ $2020 = (60)_{10}$
- ▶ $3^4 - 60 = 81 - 60 = 21 = |-X|$

Nel caso del **complemento a 2** abbiamo più semplicemente:

- ▶ si lasciano inalterate tutte le cifre fino al primo 1 compreso
- ▶ si invertono le rimanenti cifre

Complemento a 2

Esempio: Rappresentazione di -298 in complemento a 2:

- ▶ di quante cifre minimo abbiamo bisogno per rappresentare 298 ?
- ▶ $-2^{10-1} \leq 298 < 2^{10-1}$: 10 cifre
- ▶ $298 = 256 + 32 + 8 + 2$, $||298||$ in base 2 = 100101010
- ▶ $||298||$ in complemento a 2 con 10 cifre = 0100101010
- ▶ $||-298||$ in complemento a 2 con 10 cifre = 1011010110
- ▶ $||298||$ in complemento a 2 con 11 cifre = 00100101010
- ▶ $||-298||$ in complemento a 2 con 11 cifre = 11011010110

Operazioni su interi relativi in complemento a 2

Somma di due numeri

- ▶ si effettua **bit a bit**
- ▶ non è necessario preoccuparsi dei segni
- ▶ il risultato sarà corretto (in complemento a 2 se negativo)
- ▶ può verificarsi **trabocco** (overflow) \Rightarrow il risultato non è corretto

Si verifica quando il numero di bit a disposizione non è sufficiente per rappresentare il risultato.

Operazioni su interi relativi in complemento a 2

Esempio: $n = 5$, $\pm 9 \pm 3$, $\pm 9 \pm 8$

intervallo di rappresentazione: da -2^4 a $2^4 - 1$ (da -16 a 15)

rip.	00011		11001		00111		11111
+9	01001	+9	01001	-9	10111	-9	10111
+3	00011	-3	11101	+3	00011	-3	11101
+12	01100	+6	00110	-6	11010	-12	10100

In questi casi non si ha trabocco.

rip.	01000		10000
+9	01001	-9	10111
+8	01000	-8	11000
-15	10001	+15	01111
(e non +17)		(e non -17)	

Si ha **trabocco** quando il riporto sul bit di segno è diverso dall'ultimo riporto.

Operazioni su interi relativi in complemento a 2

Differenza tra due numeri: si somma al primo il complemento del secondo

Esempio: $n = 5$, intervallo di rappresentazione: da -16 a 15

rip.	11111	11001
+9	01001	01001
-9	10111	11101
<hr/>		<hr/>
0	00000	+6 00110

Numeri frazionari

- Numeri reali compresi tra 0 e 1: si rappresentano comunemente come

$$N = 0.c_{-1}c_{-2} \dots c_{-n}$$

- Il peso delle cifre dipende, al solito, dalla loro posizione e dalla base prescelta

$$N_b = c_{-1} \cdot b^{-1} + c_{-2} \cdot b^{-2} + \dots + c_{-n} \cdot b^{-n} = \sum_{i=-n}^{-1} c_i \cdot b^i$$

Esempio: Consideriamo $b = 10$ ed il numero 0.587

$$0.587_{10} = 5 \cdot 10^{-1} + 8 \cdot 10^{-2} + 7 \cdot 10^{-3}$$

Numeri frazionari

$$N_b = \sum_{i=-n}^{-1} c_i \cdot b^i \quad (\bullet)$$

- Nel caso di un numero frazionario in binario, possiamo usare la (\bullet) per convertirlo in base 10

Esempio: Convertiamo in base 10 il numero frazionario binario 0.1011_2

$$0.1011_2 = 1 \cdot 2^{-1} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = 0.6875_{10}$$

- La rappresentazione dei numeri frazionari può introdurre **approssimazioni** dovute alla limitatezza delle cifre dopo la virgola.
- L'approssimazione è comunque inferiore a b^{-n} dove n è il numero di cifre utilizzate.

Conversione di un numero frazionario da base 10 a base 2

- Il metodo più semplice consiste nell'effettuare una sequenza di moltiplicazioni per 2 prendendo ad ogni passo la parte intera del risultato come cifra binaria della rappresentazione

- **Esempio:** Convertiamo 0.125 in base 2

$$\begin{array}{rcll} 0.125 \times 2 & = & 0.25 & | 0 \\ 0.25 \times 2 & = & 0.5 & | 0 \\ 0.5 \times 2 & = & 1.0 & | 1 \end{array}$$

- In questo caso abbiamo una rappresentazione **esatta** su 3 cifre ($0.125 = 1/8$)

$$0.125_{10} = 0.001_2$$

Conversione di un numero frazionario da base 10 a base 2

- **Esempio:** Convertiamo 0.587_{10} in base 2

$$\begin{array}{rclcl}
 0.587 \times 2 & = & 1.174 & | & 1 \\
 0.174 \times 2 & = & 0.348 & | & 0 \\
 0.348 \times 2 & = & 0.696 & | & 0 \\
 0.696 \times 2 & = & 1.392 & | & 1 \\
 0.392 \times 2 & = & 0.784 & | & 0 \\
 0.784 \times 2 & = & 1.568 & | & 1 \\
 0.568 \times 2 & = & \dots & | &
 \end{array}$$

- Quindi la rappresentazione di 0.587_{10} in base 2 è:
- 0.1001_2 con 4 cifre (approssimazione accurata entro 2^{-4})
 - 0.100101_2 con 6 cifre (approssimazione accurata entro 2^{-6})

L'aritmetica reale

- L'insieme dei reali (e dei razionali) è infinito \implies non è possibile rappresentarlo tutto

Rappresentazione in virgola fissa

Si rappresentano separatamente, usando un numero fissato di cifre

- parte intera e,
- parte frazionaria

(si usa una virgola per separare le due parti)

$$N_b = c_{n-1} c_{n-2} \dots c_1 c_0 , c_{-1} c_{-2} \dots c_{-m}$$

rappresenta il numero

$$N = c_{n-1} \cdot b^{n-1} + \dots + c_0 \cdot b^0 + c_{-1} \cdot b^{-1} + \dots + c_{-m} \cdot b^{-m}$$

L'aritmetica reale

► Limitazioni della rappresentazione:

- k bit per la parte intera $\Rightarrow (-2^k, 2^k)$
- m bit per la parte frazionaria \Rightarrow precisione $\leq 2^{-m}$

Rappresentazione in virgola mobile (floating point)

Utilizza la notazione **esponenziale**. Si esprime il numero come prodotto di due parti

$$X = m \cdot b^e$$

Esempio:

$$1150 = 1.15 \times 10^3$$

ma anche

$$1150 = 0.115 \times 10^4$$

Rappresentazione in virgola mobile

Rappresentazione in **forma normalizzata** in base b

$$X = m \cdot b^e$$

- e è la **caratteristica** in base b di X : intero relativo
- m è la **mantissa** in base b di X : numero frazionario tale che $1/b \leq |m| < 1$

► Esempio:

$$1150 = \underset{\text{mantissa}}{0.115} \times \underset{\text{caratteristica}}{10^4}$$

Rappresentazione in virgola mobile

- Se la caratteristica è rappresentata dalla sequenza di cifre

$$c_1 \ c_2 \ c_3 \ \dots$$

allora rappresenta il valore

$$c_1 \cdot b^{-1} + c_2 b^{-2} + \dots$$

Esempio: $X = (5)_{10} = (101)_2$ che normalizzato diventa:

$$\begin{aligned} m &= |m| = (0.101 \dots 0000)_2 \\ e &= (11)_2 \end{aligned}$$

Rappresentazione in virgola mobile

- Fissati:
 - k bit per mantissa
 - h bit per caratteristica
 - 1 bit per il segno

l'**insieme di reali rappresentabili** è fissato (e limitato)

$$\begin{aligned} (0.1) \quad 1/2 &\leq |m| \leq \sum_{i=1}^k 2^{-i} \quad (0.11 \dots 1) \\ |e| &\leq 2^{h-1} - 1 \end{aligned}$$

- Questo fissa anche massimo e minimo (in valore assoluto) numero rappresentabile.
- Assunzione realistica: reali rappresentati con 32 bit:
 - 24 bit per la mantissa
 - 7 bit per la caratteristica (in complemento)
 - 1 bit per il segno della mantissa (0 positivo, 1 negativo)

Rappresentazione in virgola mobile

Insieme \mathcal{F} dei numeri rappresentabili in virgola mobile

- ▶ sottoinsieme finito dei numeri razionali rappresentabili
- ▶ simmetrico rispetto allo 0
- ▶ gli elementi **non** sono uniformemente distribuiti sull'asse reale
 - ▶ densi intorno allo 0

$$m_1 = 0.10, m_2 = 0.11, e_1 = -5$$

$$X_1 = 0.10 \times 10^5 = 0.0000010$$

$$X_2 = 0.11 \times 10^5 = 0.0000011$$
 - ▶ radi intorno al massimo rappresentabile

$$m_1 = 0.10, m_2 = 0.11, e_2 = 5$$

$$X_1 = 0.10 \times 10^5 = 10000$$

$$X_2 = 0.11 \times 10^5 = 11000$$
- ▶ molti razionali non appartengono ad \mathcal{F} (ed es. $1/3$, $1/5$, ...)
- ▶ non è chiuso rispetto ad addizioni e moltiplicazioni
- ▶ per rappresentare un reale X si sceglie l'elemento di \mathcal{F} più vicino ad X
- ▶ la funzione che associa ad un reale X l'elemento di \mathcal{F} più vicino ad X è detta **funzione di arrotondamento**

Limitazioni aritmetiche

Dovute al fatto che il numero di bit usati per rappresentare un numero è limitato

- ▶ perdita di precisione
- ▶ **arrotondamento**: mantissa non sufficiente a rappresentare tutte le cifre significative del numero
- ▶ **errore di overflow**: caratteristica non sufficiente (numero troppo grande)
- ▶ **errore di underflow**: numero troppo piccolo viene rappresentato come 0

Formati standard proposti da IEEE (Institute of Electrical and Electronics Engineers)

- ▶ **singola precisione**: 32 bit
- ▶ **doppia precisione**: 64 bit
- ▶ **quadrupla precisione**: 128 bit